1. Counting missing values

Sports clothing and athleisure attire is a huge industry, worth approximately [$193 billion in 2021 (https://www.statista.com/statistics/254489/total-revenue-of-the-global-sports-apparel-market/)](https://www.statista.com/statistics/254489/total-revenue-of-the-global-sports-apparel-market/) with a strong growth forecast over the next decade!

In this notebook, we play the role of a product analyst for an online sports clothing company. The company is specifically interested in how it can improve revenue. We will dive into product data such as pricing, reviews, descriptions, and ratings, as well as revenue and website traffic, to produce recommendations for its marketing and sales teams.

The database provided to us, `sports`, contains five tables, with `product_id` being the primary key for all of them:

`info`

column	data type	description
`product_name`	`varchar`	Name of the product
`product_id`	`varchar`	Unique ID for product
`description`	`varchar`	Description of the product

`finance`

column	data type	description
`product_id`	`varchar`	Unique ID for product
`listing_price`	`float`	Listing price for product
`sale_price`	`float`	Price of the product when on sale
`discount`	`float`	Discount, as a decimal, applied to the sale price
`revenue`	`float`	Amount of revenue generated by each product, in US dollars

`reviews`

column	data type	description
`product_name`	`varchar`	Name of the product
`product_id`	`varchar`	Unique ID for product
`rating`	`float`	Product rating, scored from `1.0` to `5.0`
`reviews`	`float`	Number of reviews for the product

`traffic`

column	data type	description
`product_id`	`varchar`	Unique ID for product
`last_visited`	`timestamp`	Date and time the product was last viewed on the website

brands

column	data type	description
product_id	varchar	Unique ID for product
brand	varchar	Brand of the product

We will be dealing with missing data as well as numeric, string, and timestamp data types to draw insights about the products in the online store. Let's start by finding out how complete

In [187]:

```
%%sql
postgresql:///sports
select count(*) as total_rows, count(description) as count_description,
       count(listing_price) as count_listing_price,
       count(last_visited) as count_last_visited
from info
inner join finance on info.product_id = finance.product_id
inner join traffic on finance.product_id = traffic.product_id
```

```
1 rows affected.
```

Out[187]:

total_rows	count_description	count_listing_price	count_last_visited
3179	3117	3120	2928

2. Nike vs Adidas pricing

We can see the database contains 3,179 products in total. Of the columns we previewed, only one — `last_visited` — is missing more than five percent of its values. Now let's turn our attention to pricing.

How do the price points of Nike and Adidas products differ? Answering this question can help us build a picture of the company's stock range and customer market. We will run a query to produce a distribution of the `listing_price` and the count for each price, grouped by `brand` .

In [189]:
```
%%sql
select brand, cast(listing_price as integer), count(*)
from finance
inner join brands on finance.product_id = brands.product_id
where listing_price > 0
group by brand, listing_price
order by listing_price desc
```

```
 * postgresql:///sports
77 rows affected.
```

Out[189]:

brand	listing_price	count
Adidas	300	2
Adidas	280	4
Adidas	240	5
Adidas	230	8
Adidas	220	11
Nike	200	1
Adidas	200	8
Nike	190	2
Adidas	190	7
Nike	180	4

3. Labeling price ranges

It turns out there are 77 unique prices for the products in our database, which makes the output of our last query quite difficult to analyze.

Let's build on our previous query by assigning labels to different price ranges, grouping by `brand` and `label`. We will also include the total `revenue` for each price range and `brand`.

In [191]:
```
%%sql
select brand, count(*), sum(revenue) as total_revenue,
       case when listing_price < 42 then 'Budget'
            when listing_price >= 42 and listing_price < 74 then 'Average'
            when listing_price >= 74 and listing_price < 129 then 'Expensive
            else 'Elite' end as price_category
from finance
inner join brands on finance.product_id = brands.product_id
where brand is not null
group by brand, price_category
order by total_revenue desc
```

```
 * postgresql:///sports
8 rows affected.
```

Out[191]:

brand	count	total_revenue	price_category
Adidas	849	4626980.069999999	Expensive
Adidas	1060	3233661.060000001	Average
Adidas	307	3014316.8299999987	Elite
Adidas	359	651661.1200000002	Budget
Nike	357	595341.0199999992	Budget
Nike	82	128475.59000000003	Elite
Nike	90	71843.15000000004	Expensive
Nike	16	6623.5	Average

4. Average discount by brand

Interestingly, grouping products by brand and price range allows us to see that Adidas items generate more total revenue regardless of price category! Specifically, `"Elite"` Adidas products priced $129 or more typically generate the highest revenue, so the company can potentially increase revenue by shifting their stock to have a larger proportion of these products!

Note we have been looking at `listing_price` so far. The `listing_price` may not be the price that the product is ultimately sold for. To understand `revenue` better, let's take a look at the `discount`, which is the percent reduction in the `listing_price` when the product is actually sold. We would like to know whether there is a difference in the amount of `discount` offered between brands, as this could be influencing `revenue`.

In [193]:
```
%%sql
select brand, (avg(discount) * 100) as average_discount
from finance
inner join brands on finance.product_id = brands.product_id
where brand is not null
group by brand
order by average_discount
```

```
 * postgresql:///sports
2 rows affected.
```

Out[193]:

brand	average_discount
Nike	0.0
Adidas	33.452427184465606

5. Correlation between revenue and reviews

Strangely, no `discount` is offered on Nike products! In comparison, not only do Adidas products generate the most revenue, but these products are also heavily discounted!

To improve revenue further, the company could try to reduce the amount of discount offered on Adidas products, and monitor sales volume to see if it remains stable. Alternatively, it could try offering a small discount on Nike products. This would reduce average revenue for these products, but may increase revenue overall if there is an increase in the volume of Nike products sold.

Now explore whether relationships exist between the columns in our database. We will check the strength and direction of a correlation between `revenue` and `reviews` .

In [195]:
```
%%sql
select corr(reviews, revenue) as review_revenue_corr
from reviews
inner join finance on reviews.product_id = finance.product_id
```

```
 * postgresql:///sports
1 rows affected.
```

Out[195]:

review_revenue_corr
0.6518512283481301

6. Ratings and reviews by product description length

Interestingly, there is a strong positive correlation between `revenue` and `reviews` . This means, potentially, if we can get more reviews on the company's website, it may increase sales of those items with a larger number of reviews.

Perhaps the length of a product's `description` might influence a product's `rating` and `reviews` — if so, the company can produce content guidelines for listing products on their website and test if this influences `revenue` . Let's check this out!

In [197]:
```
%%sql
select trunc(length(description), -2) as description_length,
       round(avg(cast(rating as numeric)), 2) as average_rating
from info
inner join reviews on info.product_id = reviews.product_id
where description is not null
group by description_length
order by description_length
```

```
 * postgresql:///sports
7 rows affected.
```

Out[197]:

description_length	average_rating
0	1.87
100	3.21
200	3.27
300	3.29
400	3.32
500	3.12
600	3.65

7. Reviews by month and brand

Unfortunately, there doesn't appear to be a clear pattern between the length of a product's `description` and its `rating` .

As we know a correlation exists between `reviews` and `revenue` , one approach the company could take is to run experiments with different sales processes encouraging more reviews from customers about their purchases, such as by offering a small discount on future purchases.

Let's take a look at the volume of `reviews` by month to see if there are any trends or gaps we can look to exploit.

In [199]:
```
%%sql
select brand, extract(month from last_visited) as month,
       count(reviews.*) as num_reviews
from brands
inner join traffic on brands.product_id = traffic.product_id
inner join reviews on traffic.product_id = reviews.product_id
group by brand, month
having brand is not null and extract(month from last_visited) is not null
order by brand, month
```

```
 * postgresql:///sports
24 rows affected.
```

Out[199]:

brand	month	num_reviews
Adidas	1	253
Adidas	2	272
Adidas	3	269
Adidas	4	180
Adidas	5	172
Adidas	6	159
Adidas	7	170
Adidas	8	189
Adidas	9	181
Adidas	10	192
Adidas	11	150
Adidas	12	190
Nike	1	52
Nike	2	52
Nike	3	55
Nike	4	42
Nike	5	41
Nike	6	43
Nike	7	37
Nike	8	29
Nike	9	28
Nike	10	47
Nike	11	38
Nike	12	35

8. Footwear product performance

Looks like product reviews are highest in the first quarter of the calendar year, so there is scope to run experiments aiming to increase the volume of reviews in the other nine months!

So far, we have been primarily analyzing Adidas vs Nike products. Now, let's switch our attention to the type of products being sold. As there are no labels for product type, we will create a Common Table Expression (CTE) that filters `description` for keywords, then use

In [201]:

```
%%sql
with footwear as (select description, revenue
                  from info
                  inner join finance on info.product_id = finance.product_id
                  where (description ilike '%shoe%' or
                         description ilike '%trainer%' or
                         description ilike '%foot%') and
                         description is not null)

select count(*) as num_footwear_products,
       percentile_disc(0.5) within group (order by revenue) as median_footwe
from footwear
```

```
 * postgresql:///sports
1 rows affected.
```

Out[201]:

num_footwear_products	median_footwear_revenue
2700	3118.36

9. Clothing product performance

Recall from the first task that we found there are 3,117 products without missing values for `description`. Of those, 2,700 are footwear products, which accounts for around 85% of the company's stock. They also generate a median revenue of over $3000 dollars!

This is interesting, but we have no point of reference for whether footwear's `median_revenue` is good or bad compared to other products. So, for our final task, let's examine how this differs to clothing products. We will re-use `footwear`, adding a filter afterward to count the number of products and `median_revenue` of products that are not in `footwear`.

In [203]:

```
%%sql
with footwear as (select description, revenue
                  from info
                  inner join finance on info.product_id = finance.product_id
                  where (description ilike '%shoe%' or
                         description ilike '%trainer%' or
                         description ilike '%foot%') and
                         description is not null)

select count(*) as num_clothing_products,
       percentile_disc(0.5) within group (order by revenue) as median_cloth
from info
inner join finance on info.product_id = finance.product_id
where info.description not in (select description from footwear)
```

```
 * postgresql:///sports
1 rows affected.
```

Out[203]:

num_clothing_products	median_clothing_revenue
417	503.82